

Diseño, Implementación y Utilización de una Plataforma para desarrollar Sistemas de Recuperación de Información Estructurados

Alfonso E. Romero

`aromero@correo.ugr.es`

`http://alfonso.2ya.com`

Seminario impartido en el Dep. de Ciencias de la Computación e I.A.

ETS Ingeniería Informática

Universidad de Granada

20 de Octubre de 2005

Resumen

- 1 Planteamiento del problema
 - Sistemas de Recuperación de Información Estructurada
 - Recuperación de Información Estructurada con RR.BB.

Resumen

- 1 Planteamiento del problema
 - Sistemas de Recuperación de Información Estructurada
 - Recuperación de Información Estructurada con RR.BB.
- 2 Implementación de una plataforma genérica para SRIE
 - Introducción
 - Sobre la implementación y diseño
 - Ejemplo de uso: Shakespeare
 - Desarrollo en Garnata
 - Estadísticas

Resumen

- 1 Planteamiento del problema
 - Sistemas de Recuperación de Información Estructurada
 - Recuperación de Información Estructurada con RR.BB.
- 2 Implementación de una plataforma genérica para SRIE
 - Introducción
 - Sobre la implementación y diseño
 - Ejemplo de uso: Shakespeare
 - Desarrollo en Garnata
 - Estadísticas
- 3 Referencias

Planteamiento del problema

Información estructurada

¿Por qué es tan importante la *información estructurada*?

- Información estructurada es “información que se ha analizado” [5]
- Añade más contenido a la información propiamente en sí
- Presente en multitud de aplicaciones reales:
 - **Información científica:** Abstract, Secciones, Subsecciones, Figuras, Bibliografía, . . .
 - **Sesiones que siguen una cierta estructura:** Sesiones de parlamentos, transcripción de juicios, boletines oficiales, . . .
 - Cualquier tipo de información que se revise y traduzca a un dominio estructurado (información médica, colecciones bibliográficas, . . .).

Planteamiento del problema

Sistemas de Recuperación de Información Estructurada: introducción

Particularidades que presentan los SRIE [1] sobre los SRI “clásicos”:

- De *documento a unidad estructural*.
- Relaciones de inclusión entre unidades (unidades *finales* y *contenedoras*).
- Generalmente, colecciones basadas en **XML** (estándar en texto estructurado).
- Alto número de unidades estructurales (¡en cada archivo puede haber cientos o miles!).

También pueden hacer la búsqueda más fácil para el usuario pudiendo recibir *preferencias* sobre las unidades estructurales a devolver.

⇒ Problema: **implementación** eficiente costosa.

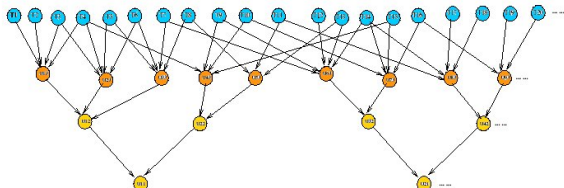
Planteamiento del problema

Desarrollo de SRIE con MGPS en este grupo de investigación I

- Modelo de RI estructurada basado en Diagramas de Influencia (CID) [2], [3], partiendo de otro anterior basado en Redes Bayesianas (BNR-SD) [4] (el implementado).
- *Nodos* para los términos (T_i) y unidades estructurales (U_j). Valores para cada variable: relevante/no relevante (t_i^+ , t_i^-). Dos tipos de unidades estructurales (finales y contenedoras). *Relaciones causales*, desde cada término a las unidades finales donde aparece, y de cada unidad (final o contenedora) a la *única* unidad que la contiene.
- No hay que almacenar *distribuciones de probabilidad*: evaluación mediante un *modelo canónico* (muy eficiente). Necesita precálculo de un conjunto de pesos (“importancia” de cada padre en su hijo).

Planteamiento del problema

Desarrollo de SRIE con MGPS en este grupo de investigación I



- Inferencia: $p(u_j^+ | pa(U_j)) = \sum_{T_i \in R(pa(U_j))} w(T_i, U_j)$,
 $p(u_k^+ | pa(U_k)) = \sum_{U_j \in R(pa(U_k))} w(U_j, U_k)$.
- Mientras $\sum_{T_i \in U_j} w(T_i, U_j) \leq 1$ y $\forall i, j, w(T_i, U_j) \geq 0$, se permiten varios esquemas de “pesado” (generalmente basados en $tf \cdot idf$).

Planteamiento del problema

Introducción al sistema I

Características “deseables” de un sistema genérico de Recuperación de información estructurada

Plataforma para el desarrollo de SRIE estructurados, que permita trabajar con *varias colecciones*, con *distintos DTDs* (gramáticas) dentro de la misma colección, varios *esquemas de pesado* para una misma colección y que proporcione un conjunto de clases con varias *funciones útiles ya implementadas*, y que proporcione *alta eficiencia* tanto en el proceso de indexación como en el de recuperación.

Planteamiento del problema

Introducción al sistema I

Características “deseables” de un sistema genérico de Recuperación de información estructurada

Plataforma para el desarrollo de SRIE estructurados, que permita trabajar con *varias colecciones*, con *distintos DTDs* (gramáticas) dentro de la misma colección, varios *esquemas de pesado* para una misma colección y que proporcione un conjunto de clases con varias *funciones útiles ya implementadas*, y que proporcione *alta eficiencia* tanto en el proceso de indexación como en el de recuperación.

Y además...

Implementar técnicas que faciliten el proceso de evaluación de las probabilidades en el modelo anteriormente presentado

Planteamiento del problema

Introducción al sistema II

- Se ha implementado una plataforma genérica para la construcción de Sistemas de RI Estructurada (no necesariamente basados en RR. BB. o en los modelos explicados). Con las características definidas anteriormente.
- Se ha contado con la **experiencia previa** de Luis, Juan y Juanma en este tipo de sistemas.
- **Lenguaje de programación:** C++ (lenguaje compilado y facilidad).
- Se ha desarrollado pensando en la **eficiencia** desde el principio.
- **Bibliotecas necesarias:** Xerces-C++ (de `apache.org`), para procesar el XML. Multiplataforma (sólo ANSI C++).

Software implementado

Implementación: nivel físico

Área textual

- Representación interna de los documentos: *índice invertido* (vocabulario + ocurrencias). El *vocabulario* siempre se mantiene *en memoria*. Las *ocurrencias en disco*.
- Búsquedas: por identificador en el vocabulario, $O(1)$. Por cadena, $O(\log|T|)$.
- Algunas ocurrencias (las de los términos con mayores *idf* (con mayor probabilidad a priori de ser necesitadas) se almacenan en una caché (caché 1) en memoria.
- *Posiciones relativas* de cada término en la unidad.
- Clases: Term, Occurrence, OccurrenceTable, Lexicon

Software implementado

Implementación: nivel físico II

Área estructural

- Información sobre unidades recubierta por la clase `Unit`.
- Para cada unidad, información (en disco) sobre tipo de unidad, ruta XPath, identificador,...
- Con otros archivos se obtienen los identificadores de los padres de la unidad, el hijo y descendientes, hermanos,... De forma *transparente* (llamadas a métodos). Éstos (incluyendo constructor a partir del identificador) sólo implican *dos accesos directos a disco*. El 1º. en un archivo de direcciones, y el 2º a los datos propiamente dichos.
- Existe un sistema de cachés dinámicas de objetos `Unit` diseñado (por ahora no implementado), por si fuera necesario.

Software implementado

Implementación: nivel lógico

Módulo de indexación

- Se ha implementado un algoritmo limitado en memoria (evita amplio uso de la misma y problemas derivados), obteniendo cuartetas (`id_termino`, `id_unidad`, `frec.`, `ptr._lista_posiciones`), que se van almacenando en memoria y escribiendo en lotes ordenados a disco cuando se sobrepasa el tamaño máximo permitido. Posteriormente, se hace un algoritmo de ordenación externa para dichos lotes.
- Se ha utilizado un parser SAX (en vez de DOM) por su menor uso de memoria. Esto acelera la indexación.
- Puede realizarse (o no) stemming (algoritmo de Porter) y usarse cualquier lista de *stopwords*.

Software implementado

Implementación: nivel lógico

Módulo de indexación (II)

- Clases: `indexBuilder` (constructor de índice que se comunica con el parser), `XMLIndexer` y `XMLAdditional`: *wrappers* para el parser de Xerces, `Quad` (cuarteta) y `QuadPool`: almacén temporal de cuartetas en memoria y algoritmo de ordenación externo, `PositionFile`: búfer de escritura del archivo de posiciones. `OccurrenceFile` creador del archivo de ocurrencias. `Globals`: constantes globales.
- Parámetros (en `Globals`): `initialLexiconSize`, `positionFileBuffer` y `quadPoolKBytes`.
- Un nuevo desarrollo *no debería modificar el módulo de indexación*.

Software implementado

Implementación: nivel lógico

Módulo de cálculo de pesos

- Un cálculo de pesos generalmente implica la participación del $idf_i = \log N/n_i$, por lo que es imposible calcularlos a la vez que se indexa. En nuestro software se separan totalmente (incluso ejecutables diferentes).
- Se utilizan dos clases: `IndexReader` que lee del índice ya construido y `Weight` (virtual, hereda de la anterior) que proporciona una serie de funciones típicas:
 - Frecuencias de un término en una unidad: *absoluta* ó *normalizada* (la primera dividida por la mayor frecuencia en la unidad, entre 0,0 y 1,0).
 - *Idf* del término *i*.
- Todo de forma eficiente y manteniendo en caché las ocurrencias leídas, *de forma transparente al desarrollador*.

Software implementado

Implementación: nivel lógico

Módulo de cálculo de pesos (II). Los pesos se escriben de forma ordenada en un archivo de pesos y **no** se insertan en la colección (para ello existe otro programa). Para ello se realiza un proceso similar al de indexación, con tripletas (`id_term.`, `id_unidad`, `peso`) (`Triad` y `TriadPool`).

Un ejemplo: pesos en el modelo BNRSD (esquema típico $tf \cdot idf$). ¿Qué tendríamos que implementar? (esto ya está hecho) (ver archivos `WeightBNRSD.cpp` y `WeightBNRSD.h`)

- Con el `true` se notan que los pesos se normalizan (se dividen por la suma de los pesos de cada término en la unidad).
- El esquema de pesos de cada unidad en su hijo es proporcional a los términos ancestros (normalizado).

Software implementado

Implementación: nivel lógico

Módulo de recuperación

- Hasta ahora sólo se ha implementado el modelo BNRSD (`BNRSD.cpp` y `BNRSD.h`), apoyándose en la clase virtual `RetrievalModel`.
- Se carga en memoria sólo el subárbol de unidades correspondientes a los términos relevantes y a sus descendientes.
- Sólo se realizan accesos a disco directos. El único valor que se calcula en tiempo de recuperación es la probabilidad de relevancia de cada unidad.
- Clases: `Query`, `RetrievalModel`,

Software implementado

Implementación: programas de usuario I

Introducción

- Se han implementado un conjunto de ejecutables separados con el conjunto de funciones de indexación y recuperación. Esto no impide que puedan añadirse más modelos, ejecutables, opciones...
- Tras la instalación y después de ejecutar `scripts/postinstall.sh` (para Linux), se crea un espacio en el directorio de usuario (`$HOME`), donde se almacenarán los datos. Si se quiere, se puede utilizar otra ubicación (el script citado da esa opción). Para windows debe configurarse manualmente, pero funciona.
- En el espacio de usuario (`$HOME/.garnata`) no se almacenan las colecciones.

Software implementado

Implementación: programas de usuario II

Lista de programas proporcionados

- `addItem`: añade un nuevo ítem (colección ó lista de stopwords) al sistema.
- `getInfo`: obtiene información, sobre una colección concreta o sobre un índice.
- `makeIndex`: crea un índice de una colección que exista en el sistema (pesos = 0).
- `makeWeightFile`: crea un archivo de pesos.
- `insertWeightFile`: inserta un archivo de pesos.
- `delItem`: elimina un ítem (lista de stopwords, índice, colección o archivo de pesos).
- `queryIndex`: programa de consulta (sólo CO).

Software implementado

Ejemplo de uso

- Introducimos la colección en el sistema: `addItem collection shakespeare archivo_lista ruta.`
- Creamos un índice “primero”: `makeIndex shakespeare primero stopwords.txt.`
- Creamos un archivo de pesos (según esquema BNRSD): `makeWeightFile shakespeare primero pesos.`
- Insertamos el archivo de pesos: `insertWeightFile shakespeare primero pesos.`
- Consultamos el índice: `queryIndex shakespeare primero.`

Software implementado

Cuestiones generales sobre desarrollo

Líneas generales sobre el desarrollo

- Clases en mayúscula, métodos y atributos en minúscula. No notación húngara.
- Documentación en inglés (por la utilidad de la misma). Uso de doxygen.
- Orientar a objetos. Constantes globales en `Globals`.
- Hacer uso de la STL (sobre todo algoritmos), implementación eficiente.
- Probar utilizando los programas de test.
- Depurar memoria con `valgrind`.

Software implementado

Desarrollo de nuevos modelos I

¿Cómo desarrollar otros modelos de recuperación?

- 1 Heredar de la clase `RetrievalModel` (ver, por ejemplo, `BNRSD`).
- 2 Programar los métodos virtuales (depende del modelo).
- 3 Dos opciones:
 - O construir un programa de recuperación nuevo (ver `queryIndex.cpp` y `GarnataQueryIndex.cpp`).
 - O añadir el nuevo modelo a `GarnataQueryIndex.cpp`.

Software implementado

Desarrollo de nuevos modelos II

Otras tareas que pueden (y deben) hacerse (es posible que sean implementadas a corto plazo)

- Añadir un módulo (genérico) de expansión de consultas.
- Soportar algún lenguaje de consulta “estándar” (p. ej.: NEXI).
- Añadir un módulo de evaluación.
- Compresión en los índices.

Software implementado

Estadísticas del proyecto I

Código:

- **Archivos de código** (sin contar interfaz): 81.
- **Clases implementadas:** (con entidad) 47 (sin contar interfaz gráfico).

Tiempo de indexación:

- 1 **Shakespeare:** 7.6MB, Obras completas.
 - Indexación: 1'10"
 - Cálculo de pesos: 1'42"
 - Inserción de pesos: 0'25"
- 2 **INEX:** 686MB, 126 volúmenes de revistas del IEEE.
 - Indexación: 45'
 - Cálculo de pesos: 1h07'
 - Inserción de pesos: 9'22"

Software implementado

Estadísticas del proyecto II

Tamaño de los índices: (no se ha usado compresión ni stemming)

- ❶ **Shakespeare:** 28MB (400 % el tamaño de la colección)
 - Vocabulario: 776KB
 - Ocurrencias + índice directo: 8.9MB
 - Índice estructural: 13MB
- ❷ **INEX:** 1.8GB (300 % el tamaño de la colección)
 - Vocabulario: 16.5 MB (sin stemming!)
 - Ocurrencias + índice directo: 620MB
 - Índice estructural: 790MB

Software implementado

Estadísticas del proyecto II

Tiempo de consulta:

A diferencia de otros modelos de RI, documentos que no contengan todos los términos de la consulta pueden obtener un valor de probabilidad $> 0 \implies$ el tiempo de consulta se incrementa conforme lo hace el tamaño de la consulta.

- En *Shakespeare*, prácticamente cualquier consulta es inmediata.
- En *INEX*, los términos altamente comunes (p. ej. *network*) causan problemas (más de 10 minutos de procesamiento): esto puede solucionarse mostrando las unidades según van acumulando probabilidades (propagación “por niveles” en vez de “en profundidad”. Los términos más o menos infrecuentes (p. ej.: “bayesian”) funcionan bien.




Software implementado

Datos de contacto

- **Desarrollador (y mantenedor):** Alfonso E. Romero (yo).
- **Correo:** `aromero@correo.ugr.es`,
`alfonsoeromero@gmail.com`. Sugerencias (¿podrías implementar X ó Y?), Bugs (X no funciona), Opinión (X funcionaría mejor si...), Spam,...
- **Versión actual:** 0.1.
- **Localización:** en `utai.ugr.es` (por determinar).
- **Sitio web:** por determinar. Seguramente colgando de `http://alfonso.2ya.com`.

FIN
Gracias por su atención

Referencias

-  Y. Chiaramella, *Information Retrieval and Structured Documents*, Lect. Notes Comput. Sc. **1980**: 286–309, 2001.
-  L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, *Using context information in structured document retrieval: an approach based on influence diagrams*, Inform. Process. Manag. **40**(5): 829–847, 2004.
-  L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, *Improving the Context-Based Influence Diagram Model for Structured Document Retrieval: Removing Topological Restrictions and Adding New Evaluation Methods*, Lect. Notes Comput. Sc. **3408**: 215–229, 2005.



F. Crestani, L. M. de Campos, J. F. Huete, J. M. Fernández-Luna, *A Multi-layered Bayesian Network Model for Structured Document Retrieval*, Lect. Notes Comput. Sc. **2711**: 74–86, 2003.



S. J. DeRose, *Navigation, access, and control using structured information*. American Archivist, **60**(33), 298–309, 1997.